

# Eclipse Plug-in 개발방법

(version 1.0)

작성자 : 조 준래 (wnsfo@hansung.ac.kr)

© 한성대학교 컴퓨터공학과 임베디드 시스템 실험실  
© **Embedded Systems Lab. Hansung University**



이 문서는 [크리에이티브 커먼즈 코리아 저작자표시 2.0 대한민국 라이선스](https://creativecommons.org/licenses/by-nc-sa/2.0/kr/)에 따라 이용하실 수 있습니다.

## **Eclipse Plug-in Develop**

<b>Part I. Basic Information .....</b>	<b>1</b>
<b>Part II. Project .....</b>	<b>1</b>
<b>Part III. Extensions .....</b>	<b>6</b>
<b>Part IV. The Others .....</b>	<b>9</b>
<b>Part V. Example .....</b>	<b>10</b>
<b>Part VI. Full Sources .....</b>	<b>22</b>



## Part I. Basic Information

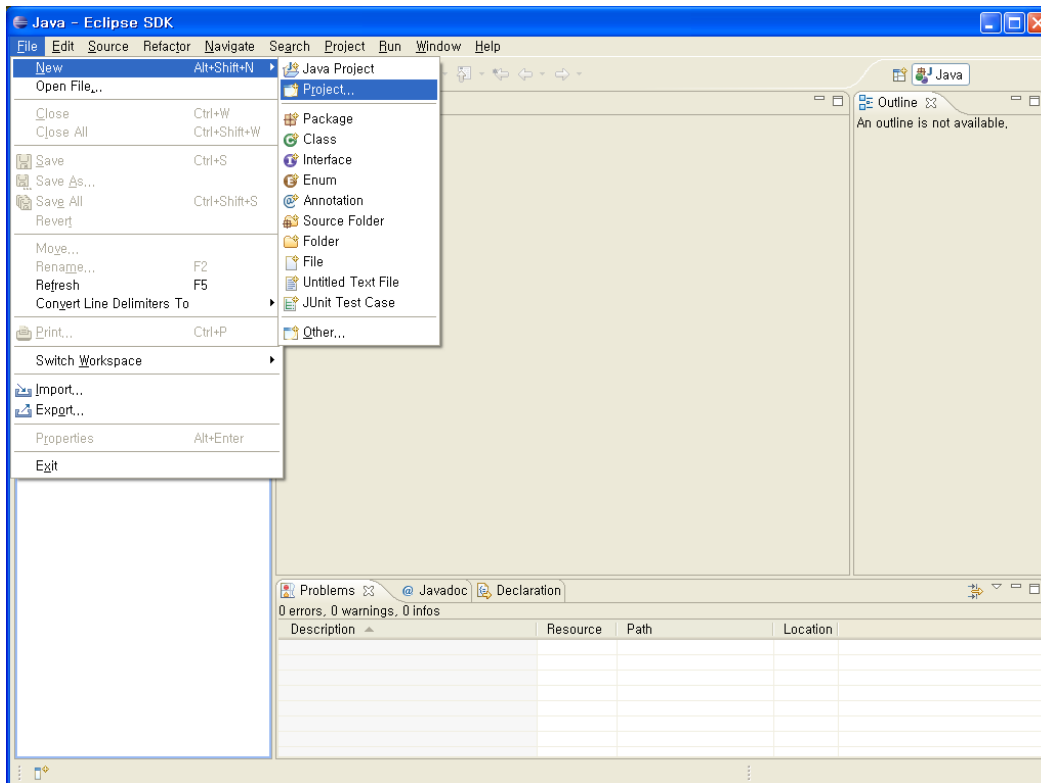
Eclipse plug-in을 개발하기 위해서 기본적으로 알아야 하는 것들은 아래와 같다.

- ▶ Eclipse plug-in을 개발하기 위해 가장 기본적으로 Eclipse가 사용되고 java로 구현하기 때문에 java를 다룰 줄 알아야 한다.
- ▶ 많은 내용을 다루지 않지만, XML에 대한 이해로 좀 더 쉬운 작업이 가능해 진다.
- ▶ 사용되는 Eclipse의 버전은 3.3 Europa 버전이다.

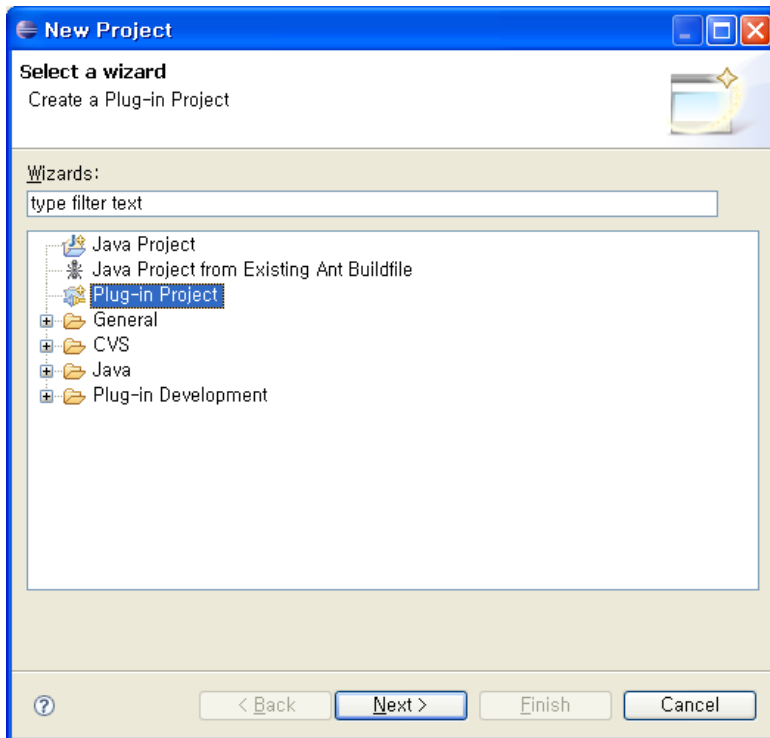
## Part II. Project

Eclipse plug-in을 개발의 시작인 프로젝트 생성에 대한 간단히 설명은 아래와 같다.

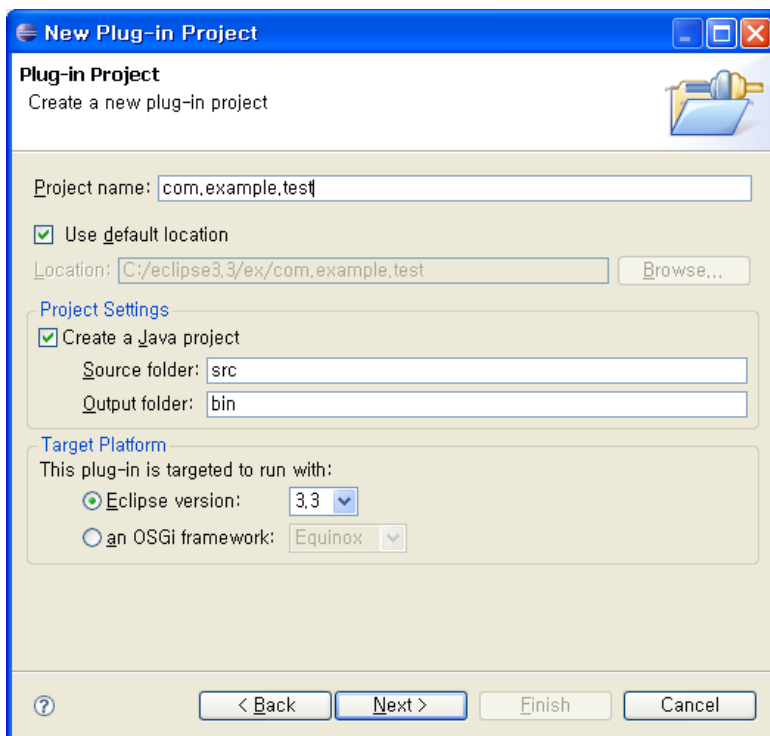
- ▶ File -> New -> Project를 클릭한다.



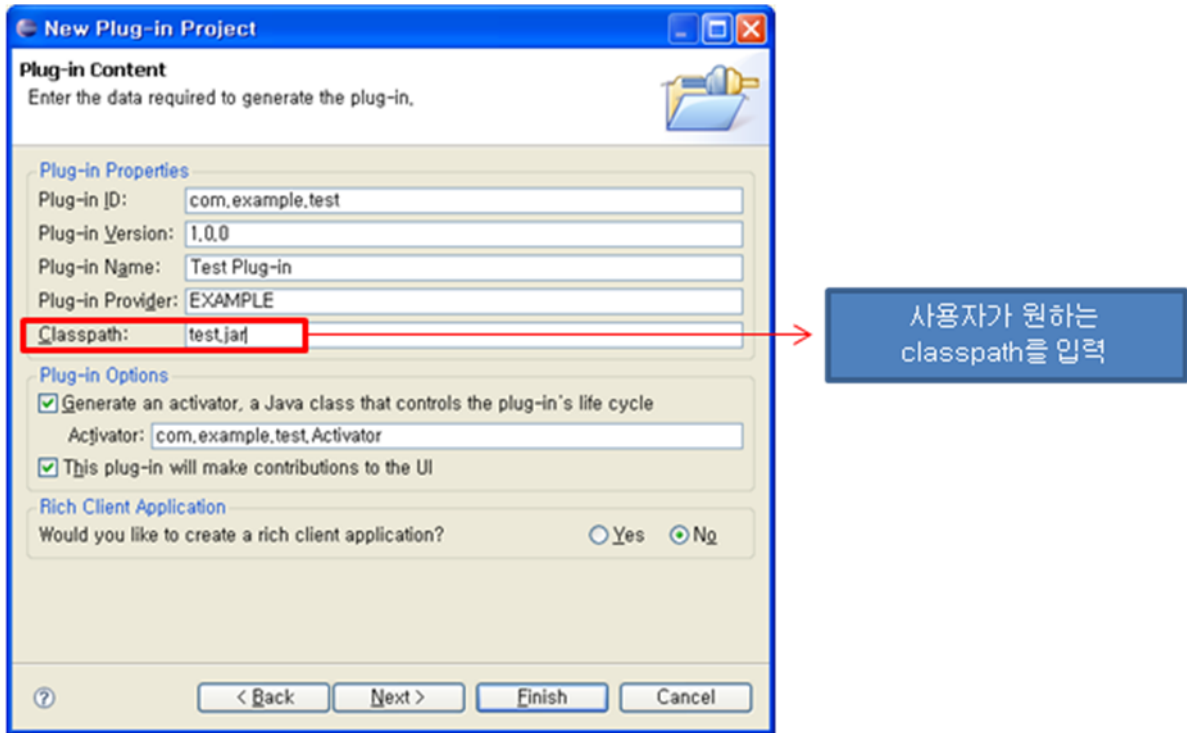
- ▶ Plug-in Project를 선택하고 Next를 클릭한다.



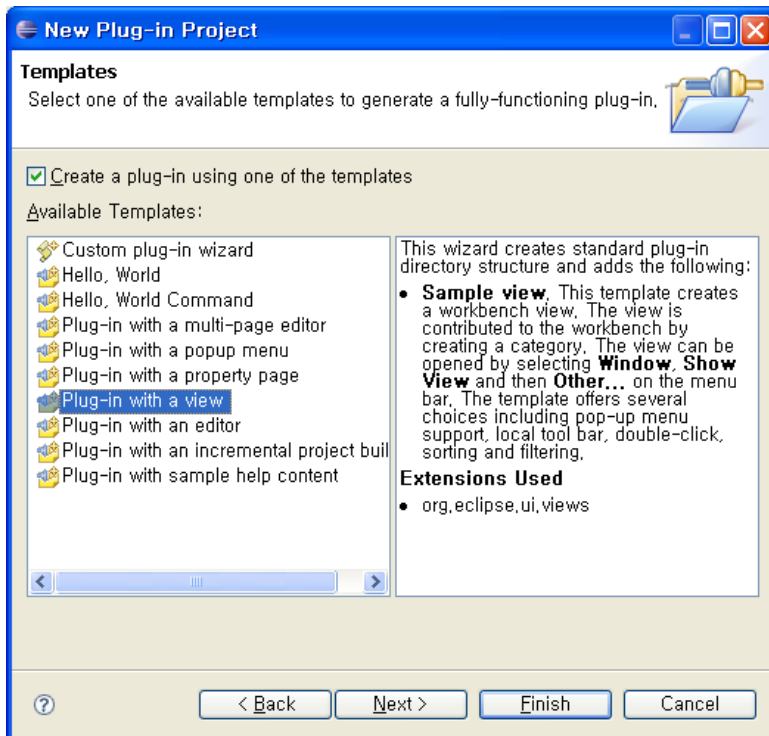
- ▶ Project name을 설정하고 Next를 클릭한다. (Eclipse 버전에 따라 화면이 다르다.)



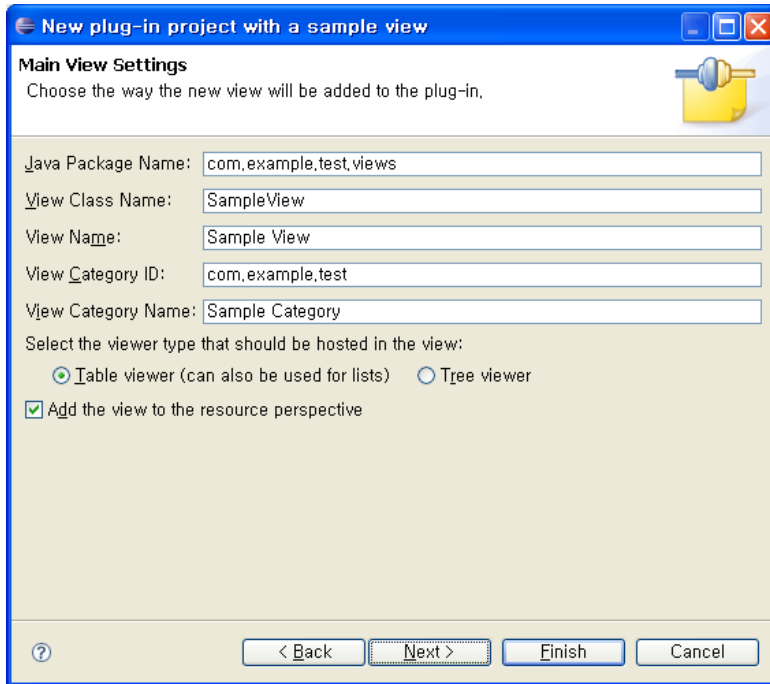
▶ Classpath를 입력하고 Next를 클릭한다.



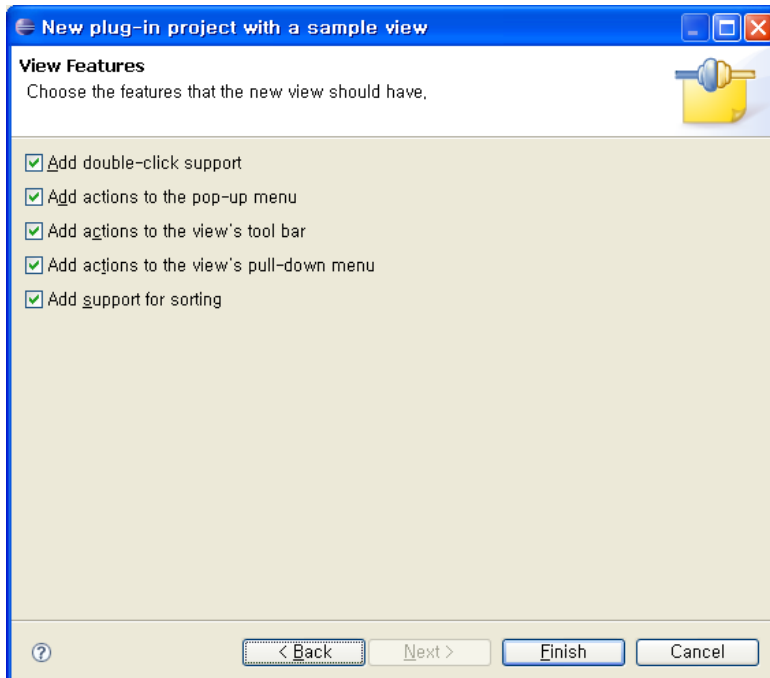
▶ 원하는 Template를 선택하고 Next를 클릭한다. (여기서는 view를 선택하였다.)



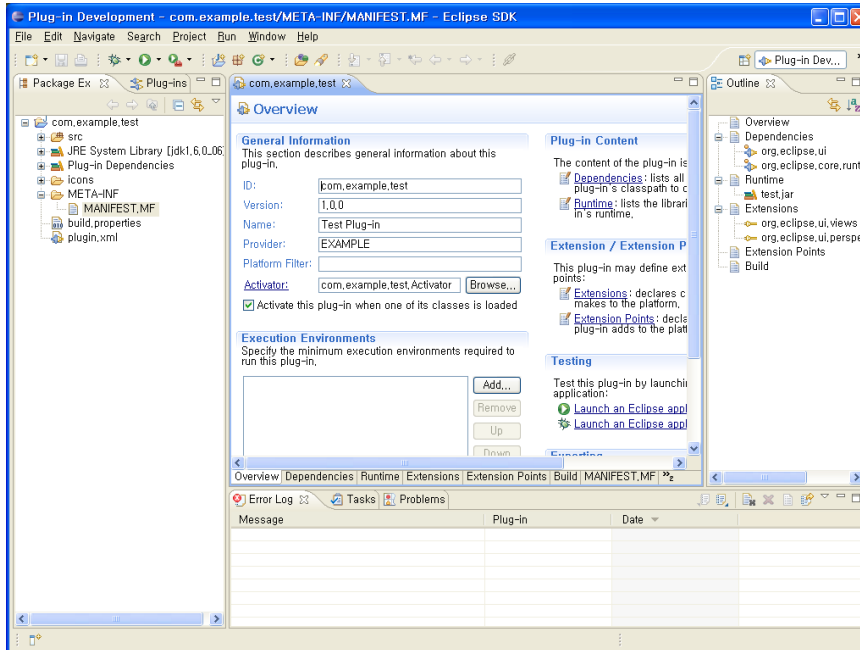
- ▶ 각 이름들을 원하는 이름으로 설정하고 **Next**를 클릭한다.



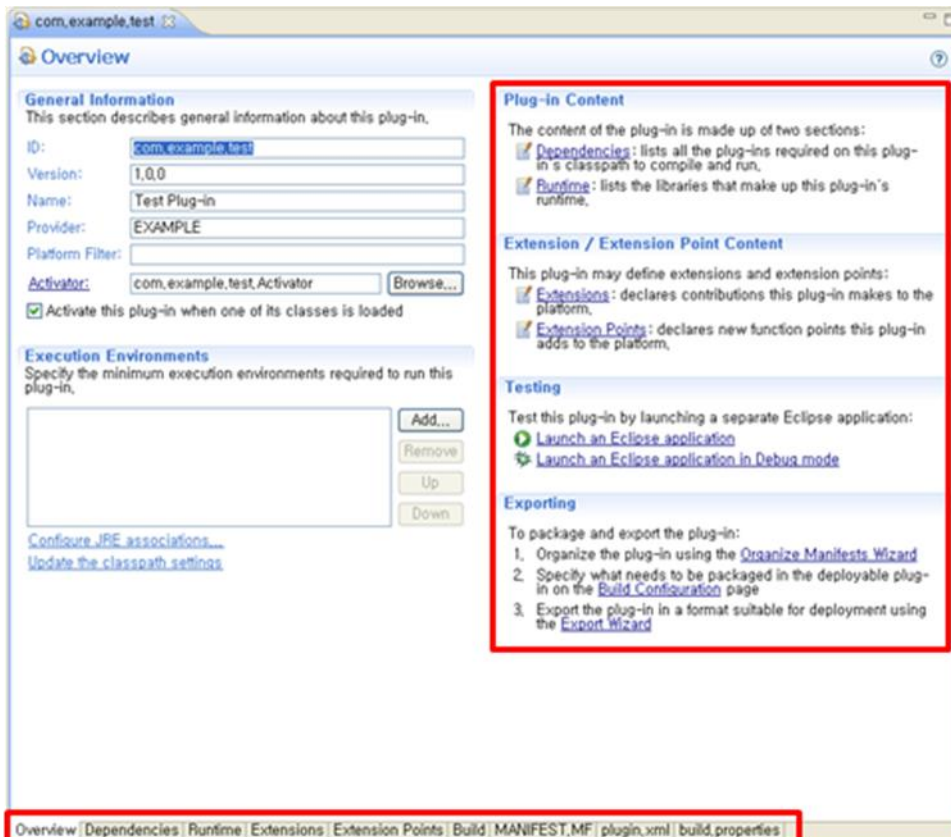
- ▶ 원하는 코드의 생성을 체크하고 **Finish**를 클릭한다.



▶ 다음과 같이 프로젝트가 생성된다.



▶ 여기에서 프로젝트를 확장하거나 변경할 수 있다. (현재 보이는 탭은 Overview)



≫ 실선 부분은 다른 설정을 하는 페이지로 이동할 수 있는 부분이다.



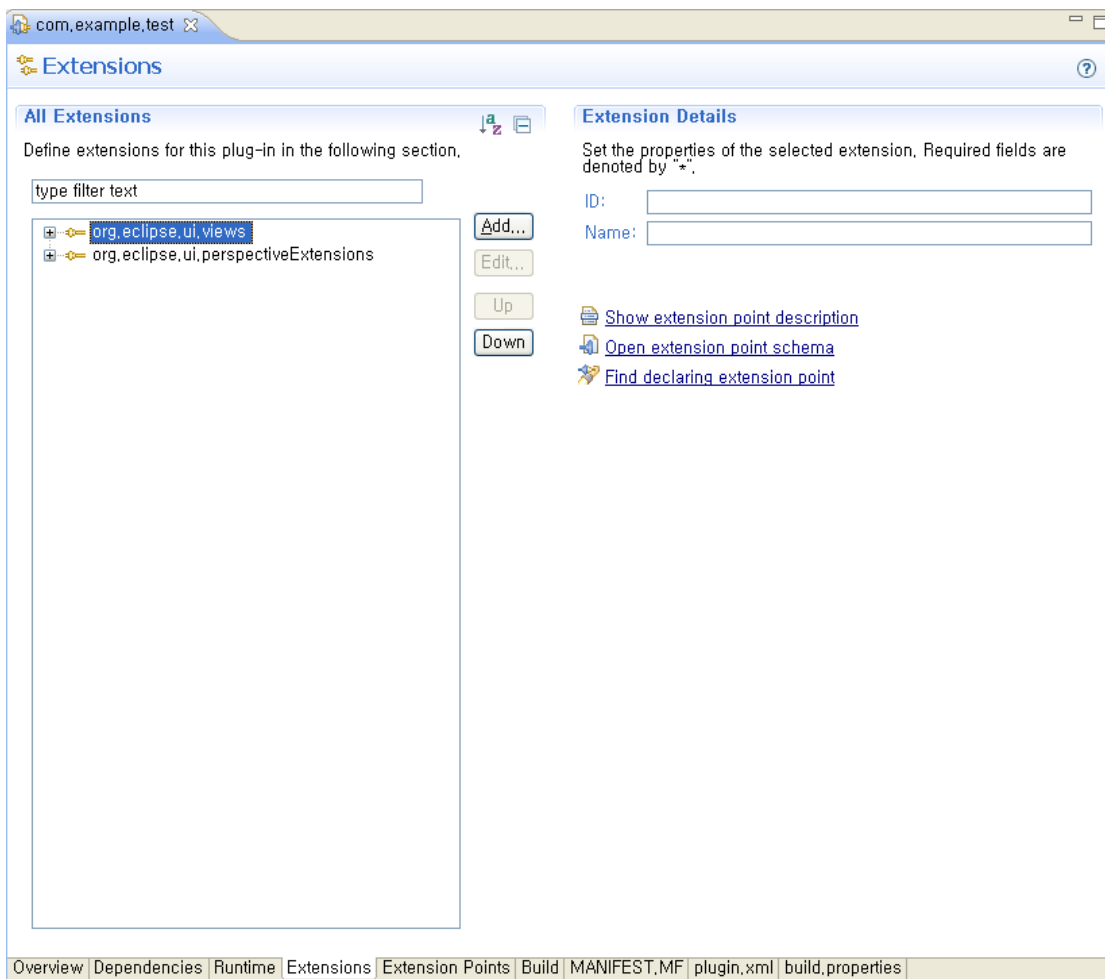
▶ 각 페이지에 대한 세부적인 선택 사항은 옵션을 확인하여 알 수 있고, 자세한 내용은 관련서적을 참고하기 바란다.

▶ 프로젝트의 확장이나 변경으로 많은 부분을 추가 변경할 수 있다. 여기에서 XML을 사용하고 Overview와 Extensions를 비롯한 많은 설정 페이지를 이용하여 쉽게 변경할 수 있다. 또한 거의 모든 설정 부분을 확인 할 수 있고, 자세한 내용은 관련 서적을 참고하기 바란다.

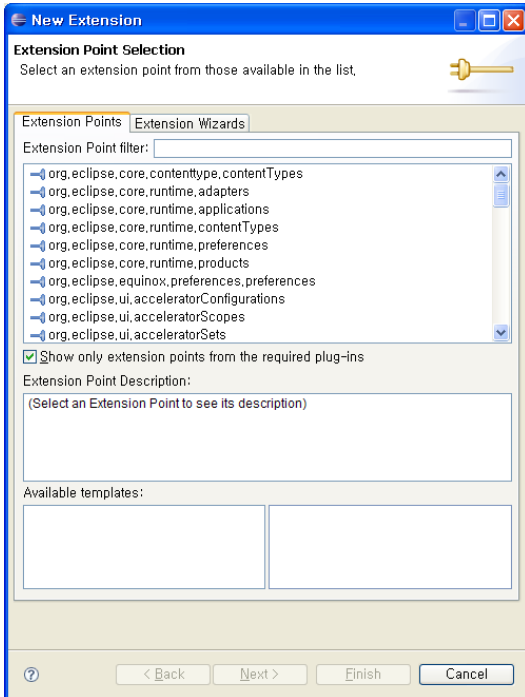
## Part III. Extensions

생성한 프로젝트에 Extension의 Template을 추가하는 방법은 아래와 같다.

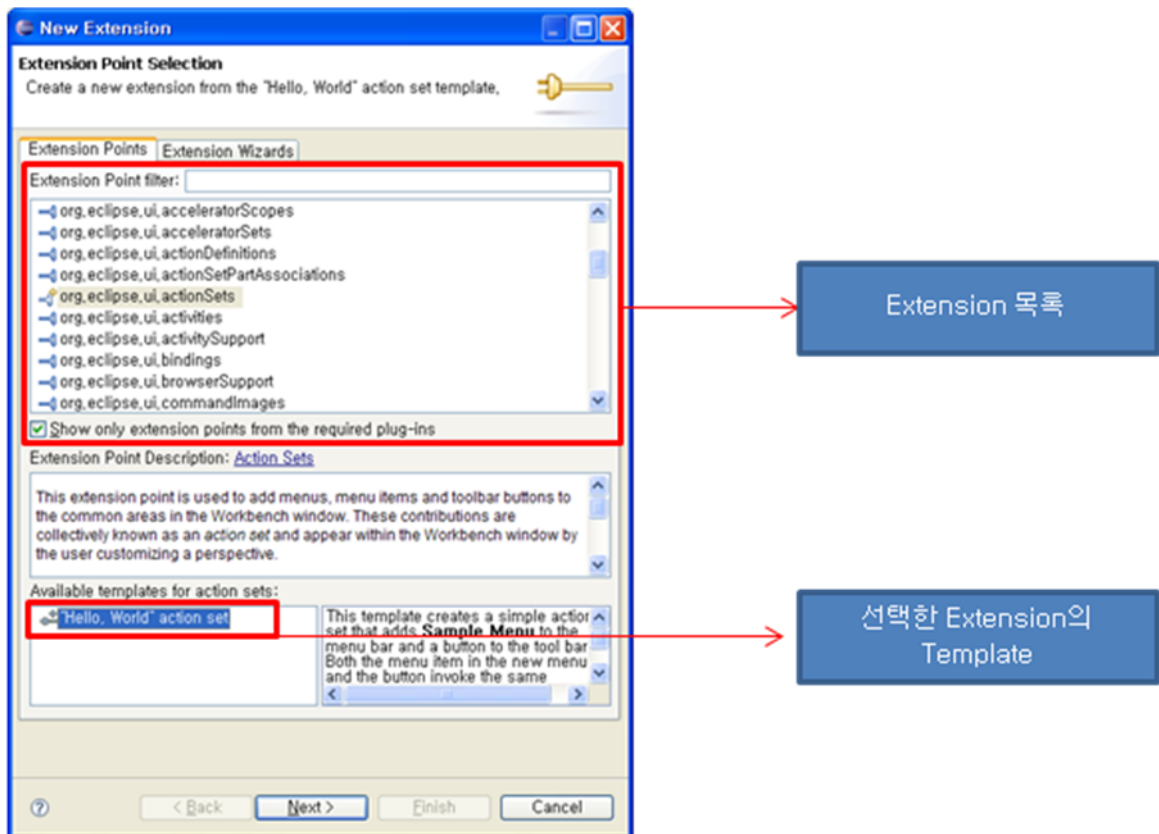
▶ 다음은 현재 Extension을 보여주고 Add로 새로운 Extension을 추가한다.



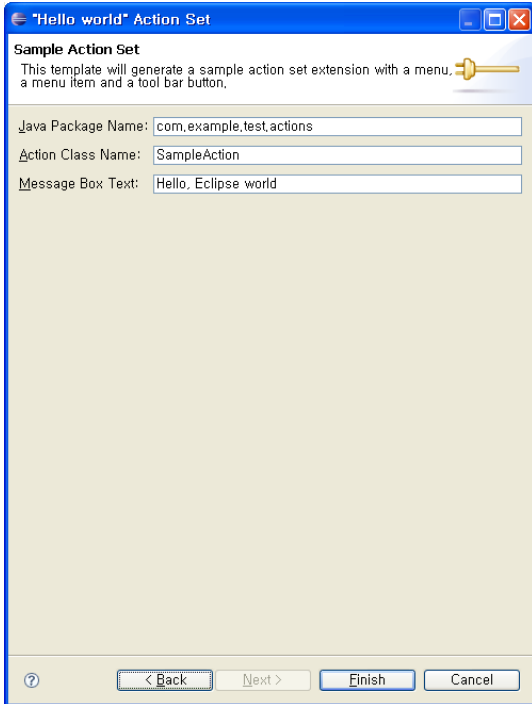
▶ Add 버튼을 클릭하면 확장 가능한 목록을 보여준다.



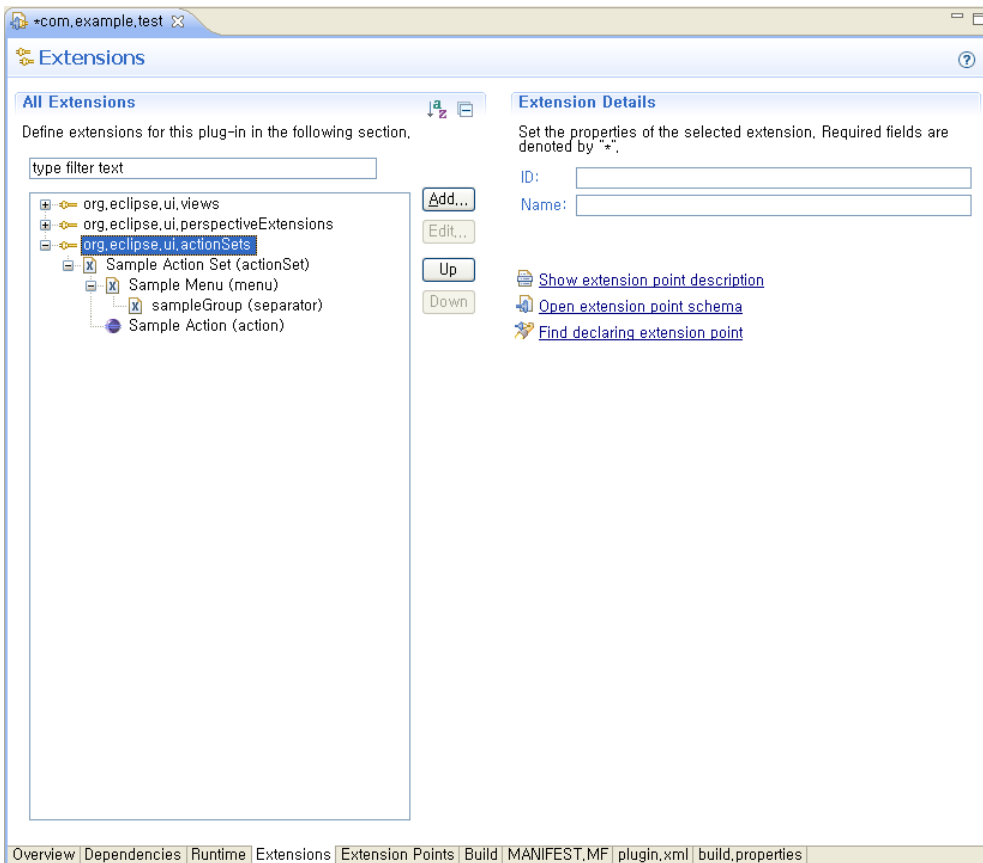
▶ 원하는 확장을 선택한 후 Next를 클릭한다. (여기서 org.eclipse.ui.actionSets의 Template를 선택)



▶ 각 이름을 확인하고 수정하여 **Finish**를 클릭한다.



▶ 선택한 **Extension**의 **Template**이 생성된다.



- ▶ **Extension**의 **Template**은 각 **Extension**의 간단한 기능을 보여주는 샘플 소스가 생성이 되어 수정할 수 있고, 만약 생성할 **Extension**에 대해 정확히 알고 있다면 **Template** 없이 생성하여 직접 작성할 수 있다.
- ▶ 각 **Extension**의 이름들은 패키지, 클래스, 출력 메시지 등과 연관되어 있고 자세한 내용은 관련서적을 참고하기 바란다.
- ▶ 각 **Extension**의 하위 관계들은 **Extension**에 따라 다르므로 사용할 **Extension**에 대하여 참고 자료를 확인하거나 **Template**를 이용하여 그 사용법을 숙지한다.
- ▶ **Extension**에는 **view**, **action** 등 다양하고 많은 기능이 있다. 구현하고 싶은 **Plug-in**의 특성을 잘 파악하여 요구사항에 맞는 **Extension**으로 구현해야 한다.

## Part IV. The Others

Eclipse plug-in을 개발하는데 사용되거나 알아 두면 좋은 것에 대해 간단히 소개하겠다.

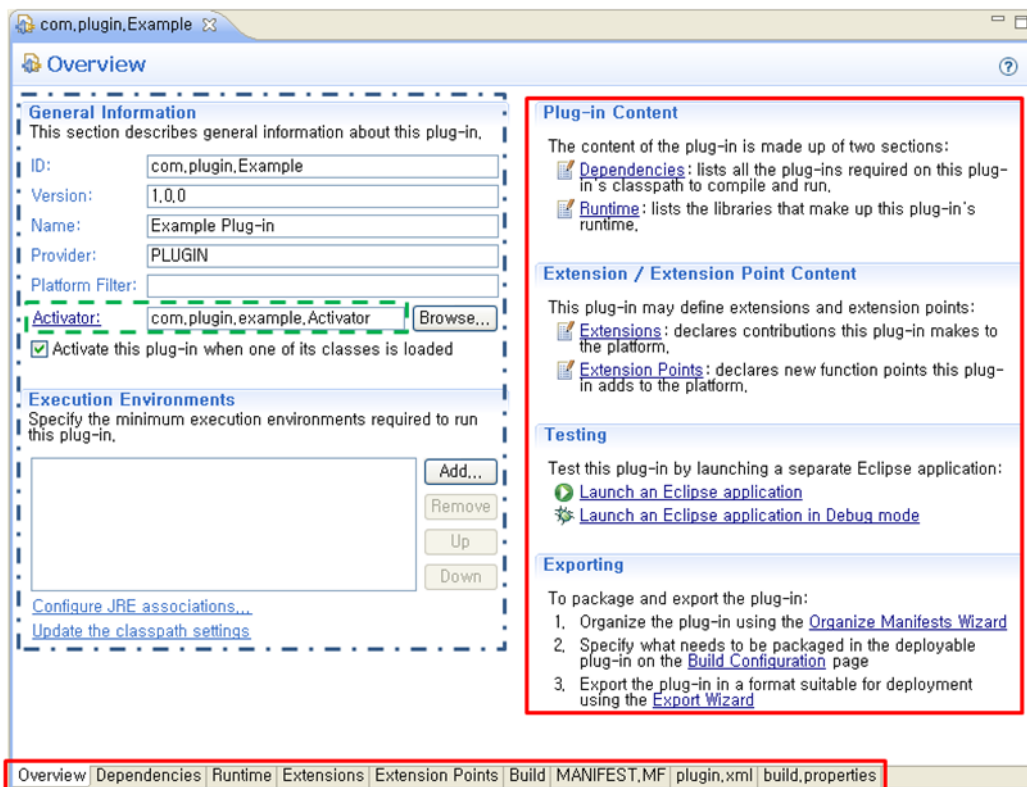
- ▶ **SWT** : 독립실행형 **Java Application**이나 **Eclipse plug-in**을 작성하는데 사용하는 **widget**을 다양하게 제공
- ▶ **JFace Viewer** : 객체 지향 데이터 사용을 위해 필요
- ▶ **Eclipse**의 구조 : **Eclipse**의 기본구조를 파악함으로써 **Eclipse**를 구성하는 **plug-in**에 대한 이해를 높임
- ▶ **Action** : 메뉴와 같은 여러 가지 **action**
- ▶ **View** : 일반적인 **view** 또는 기존 **view**를 확장
- ▶ **Editors** : **view**와 비슷하지만 다른 **Editor** 기능
- ▶ 리소스 변경 추적 : 리소스의 동기화 유지 등의 기능
- ▶ **Perspective** : **view**나 **action** 등 을 묶어서 사용할 수 있는 기능

- ▶ 대화창, 마법사 : 대화창 또는 마법사 기능
- ▶ 환경설정 페이지 : Eclipse가 제공하는 환경 페이지와 같은 방식의 환경 설정 기능
- ▶ Property : Eclipse 환경 내에 존재하는 리소스나 객체에 적용
- ▶ Export : 작성한 plug-in을 내보내는 방법
- ▶ 기타 : Eclipse plug-in을 개발하기 위해 그 밖의 많은 기능에 대하여 알아야 한다.
- ▶ 자세한 내용은 관련서적을 찾으면 어렵지 않게 확인할 수 있다.

## Part V. Example

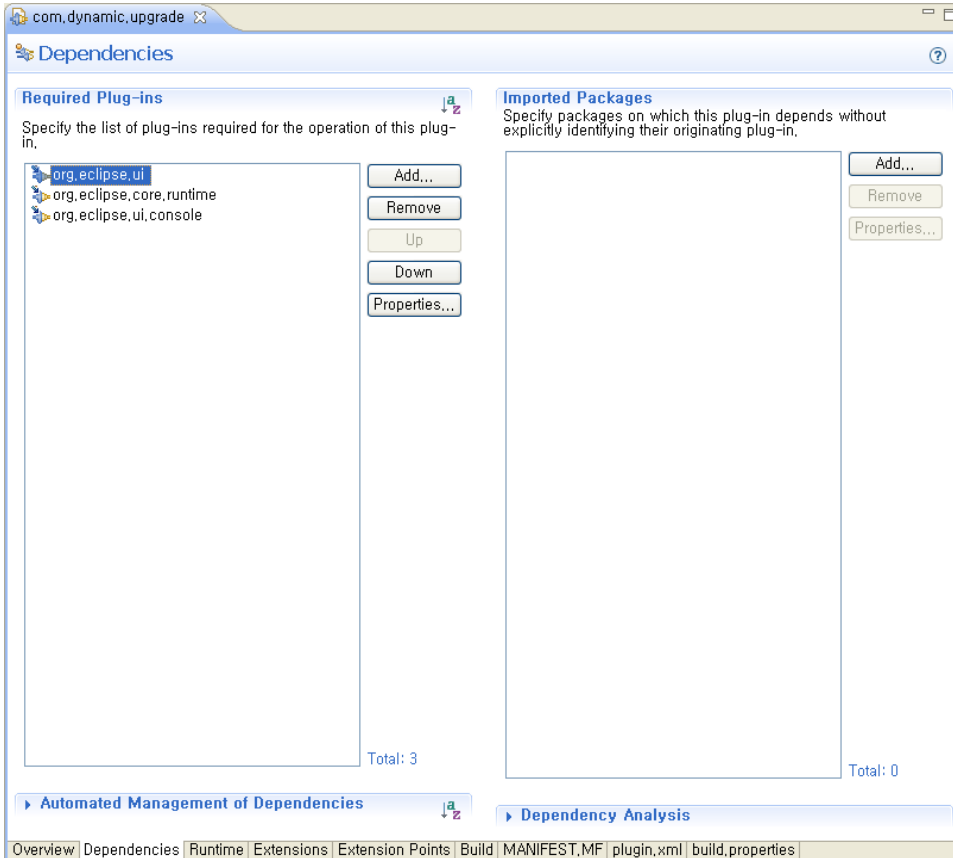
Eclipse plug-in의 간단한 예제로 단순한 계산 프로그램을 작성한 것으로 프로젝트와 필요한 Extension을 생성한 후의 개발에 대하여 살펴보도록 하겠다.

- ▶ 예제로 살펴 볼 프로젝트의 기본 정보이다.



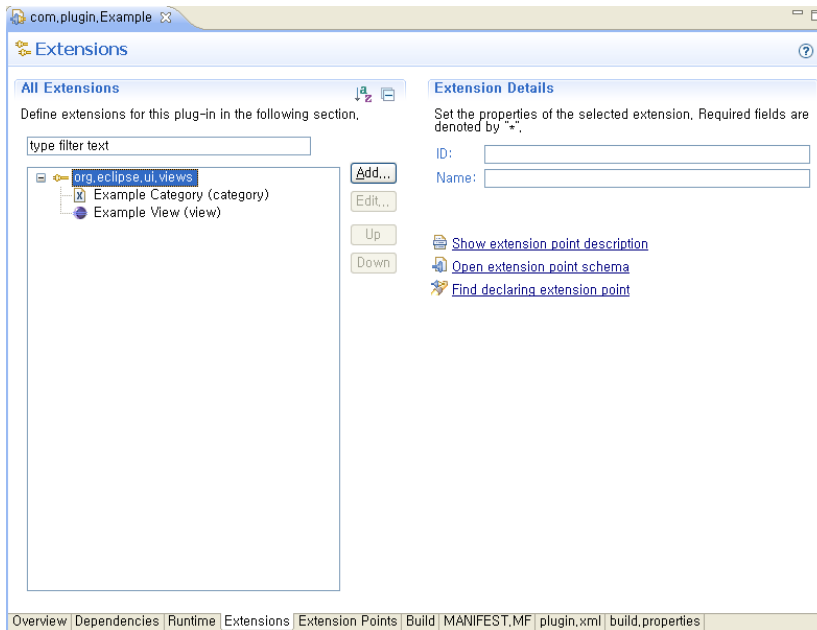
» 이 프로젝트의 ID는 `com.plugin.Example` 이고 버전은 1.0.0 이름은 `Example Plug-in`. 플러그인이 처음 로드 될 때 사용되는 `com.plugging.example.Activator` 클래스로 지정(작은 점선)하였다. (큰 점선) 실선은 각 설정으로 이동할 때 사용된다.

▶ 프로젝트가 의존적으로 사용하는 플러그인



» `org.eclipse.ui`, `org.eclipse.core.runtime`, `org.eclipse.ui.console`

▶ 프로젝트의 Extension



≫ `org.eclipse.ui.view`은 `view`를 사용하기 위한 `Extension`으로 `view`에 나타나는 이름을 `Example View`로 지정하였다.

▶ MANIFEST.MF의 내용

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Example Plug-in
Bundle-SymbolicName: com.plugin.Example; singleton:=true
Bundle-Version: 1.0.0
Bundle-ClassPath: example.jar
Bundle-Activator: com.plugin.example.Activator
Bundle-Vendor: PLUGIN
Require-Bundle: org.eclipse.ui,
org.eclipse.core.runtime,
org.eclipse.ui.console
Eclipse-LazyStart: true
```

≫ 프로젝트에 대해 간단히 정의하고 있다.

▶ **plugin.xml**의 내용

```

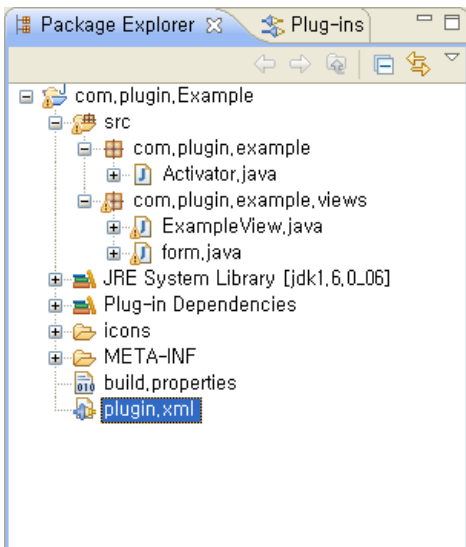
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
  <extension
    point="org.eclipse.ui.views">
    <category
      name="Example Category"
      id="com.plugin.Example">
    </category>
    <view
      name="Example View"
      icon="icons/sample.gif"
      category="com.plugin.Example"
      class="com.plugin.example.views.ExampleView"
      id="com.plugin.example.views.ExampleView">
    </view>
    </extension>
</plugin>

```

» XML을 이용하여 **Extension**의 정보에 대해서 정의하고 있다.

▶ 위 모든 내용은 이름 및 기타 정보만을 지정한 것으로 겉 모양만 정의한 것이라 할 수 있다.

▶ 현재의 프로젝트의 구조 및 동작하기 위한 클래스들이다. (필요한 클래스를 추가한 것이다.)





▶ 우선 **Activator** 클래스를 살펴 보겠다.

```
public Activator() {
    MessageConsole console = new MessageConsole("Compile Process",null);
    IConsoleManager manager = (IConsoleManager)
        ConsolePlugin.getDefault().getConsoleManager();
    manager.addConsoles(new IConsole[]{console});
    manager.showConsoleView (console);
    MessageConsoleStream stream = console.newMessageStream();
    System.setOut(new PrintStream(stream));
}
```

» 이 **class**는 프로젝트를 생성할 때 자동으로 생성되며 위에 보이는 코드는 이 클래스의 생성자로 이 **plug-in**에서 출력되는 내용을 **console**에 출력하기 위한 설정이다.

▶ **ExampleView** 클래스

```
public class ExampleView extends ViewPart {
    private form fr;
    public ExampleView() {
    }
    public void createPartControl(Composite parent) {
        Composite cont = new Composite(parent, SWT.BORDER);
        fr = new form(cont);
    }
    @Override
    public void setFocus() {
        // TODO Auto-generated method stub
    }
}
```

» 실제 **view**를 생성하는 부분으로 변수 **fr**은 각 컴포넌트와 그에 따른 동작이 작성된 클래스이다. 실제로 작성할 때 처음 코드가 생성되면 리스트나 트리 구조를 갖게 되는데 예제에서 사용될 컴포넌트를 생성하기 위해서 수정된 것이다.

## ▶ form 클래스

```
public class form {  
  
    static private Process ps;  
  
    static private BufferedReader br;  
  
    static private BufferedWriter bw;  
  
    private Group startGroup;  
  
    private Group executeGroup;  
  
    final List list;  
  
    private Button sum;  
  
    private Button mult;  
  
    final Text t1;  
  
    final Text t2;  
  
    private Button exe;  
  
    public form(Composite shell){  
  
        ps = null;  
  
        startGroup = new Group(shell, SWT.NULL);  
        startGroup.setText("Start Setting");  
        startGroup.setBounds(25, 25, 300, 80);  
  
        list = new List(startGroup, SWT.SINGLE);  
        list.setItems(new String[] {  
            "Integer", "Float"});  
  
        list.setBounds(25, 25, 50, 25);  
        list.setSelection(0);  
  
        executeGroup = new Group(shell, SWT.NULL);  
        executeGroup.setText("Execute");  
        executeGroup.setBounds(25, 120, 300, 100);  
  
        sum = new Button(executeGroup, SWT.RADIO);  
        mult = new Button(executeGroup, SWT.RADIO);  
        sum.setText("Addition");  
        mult.setText("Multiplication");  
        sum.setBounds(25, 25, 80, 25);  
        mult.setBounds(110, 25, 90, 25);  
  
        t1 = new Text(executeGroup, SWT.SINGLE);
```

```
t2 = new Text(executeGroup, SWT.SINGLE);
t1.setBounds(25, 60, 50, 15);
t2.setBounds(85, 60, 50, 15);

exe = new Button(executeGroup, SWT.PUSH);
exe.setText("Calculation");
exe.setBounds(150, 50, 90, 25);
exe.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        String str;
        try {
            if (list.getSelectionIndex() == 0) {
                str = "C://calculation.exe i";
            } else {
                str = "C://calculation.exe f";
            }
            if (sum.getSelection()) {
                str = str+" s "+t1.getText()+
                    " "+t2.getText();
            } else {
                str = str+" m "+t1.getText()+
                    " "+t2.getText();
            }
            ps = Runtime.getRuntime().exec(str);
            br = new BufferedReader(new
                InputStreamReader(ps.getInputStream()));
            bw = new BufferedWriter(new
                OutputStreamWriter(ps.getOutputStream()));
            System.out.println(br.readLine());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});
}
```

» 생성자 부분에서 `composite`을 받아서 각 컴포넌트를 생성한다. 여기서 `Group`, `List`, `Button`, `Text` 등의 컴포넌트가 사용되었고 특히 `Button`은 `push`, `radio`의 옵션으로 다른 형태를 취한다. 버튼이 눌렸을 때 예제가 동작하기 위해 `SelectionListener`를 작성하였다. `SelectionListener`에서는 C로 작성된 간단한 계산 프로그램을 `Background`로 실행하며, 선택된 옵션을 전달하여 계산한 뒤 결과를 출력한다.

▶ C로 작성된 간단한 계산 프로그램

```
#include <stdio.h>
#include <stdlib.h>

int type;

void sum(int i1, int i2, float f1, float f2) {
    if (type == 0) {
        printf("%d + %d = %d\n", i1, i2, i1+i2 );
    } else if(type == 1) {
        printf("%f + %f = %f\n", f1, f2, f1+f2 );
    }
    return ;
}

void mul(int i1, int i2, float f1, float f2) {
    if (type == 0) {
        printf("%d * %d = %d\n", i1, i2, i1*i2 );
    } else if(type == 1) {
        printf("%f * %f = %f\n", f1, f2, f1*f2 );
    }
    return ;
}

void main(int argc, char* argv[]){
    int i1, i2;
    float f1, f2;

    if (argv[1][0] == 'i') {
        type = 0;
        i1 = atoi(argv[3]);
```

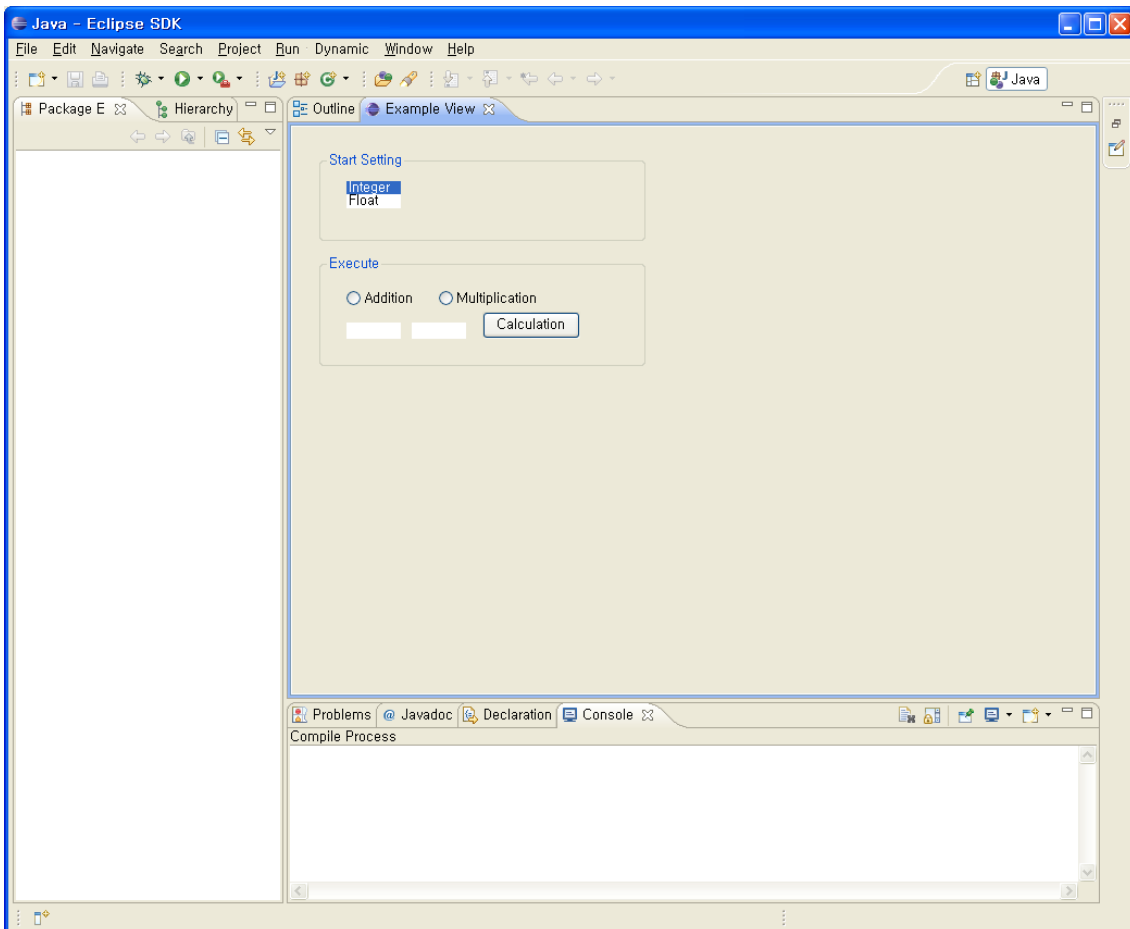
```

        i2 = atoi(argv[4]);
    } else if(argv[1][0] == 'f'){
        type = 1;
        f1 = atof(argv[3]);
        f2 = atof(argv[4]);
    }
    if ( argv[2][0] == 's' ) {
        sum(i1, i2, f1, f2);
    } else if (argv[2][0] == 'm'){
        mul(i1, i2, f1, f2);
    }
    return ;
}

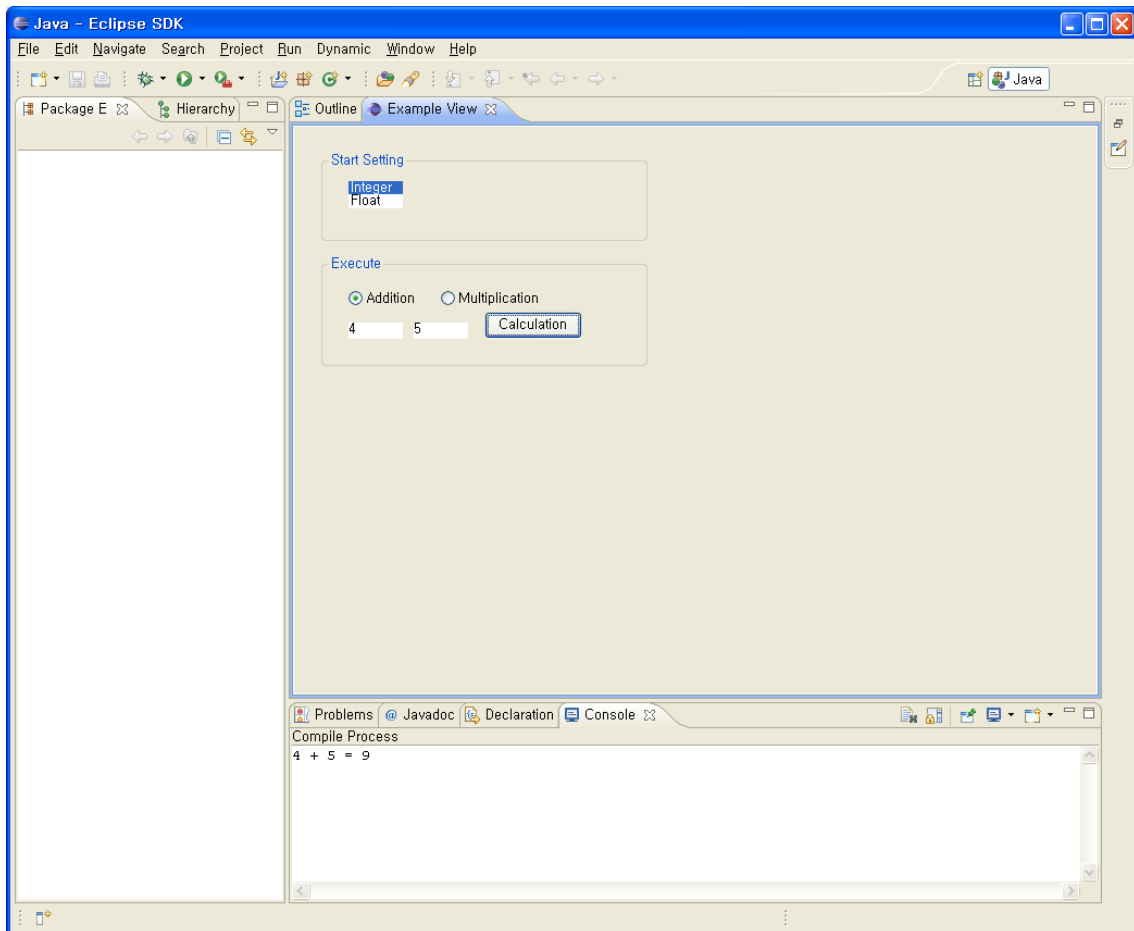
```

➤ 옵션을 받아서 계산을 하여 출력하는 프로그램이다.

▶ 개발된 plug-in을 적용한 Eclipse의 모습



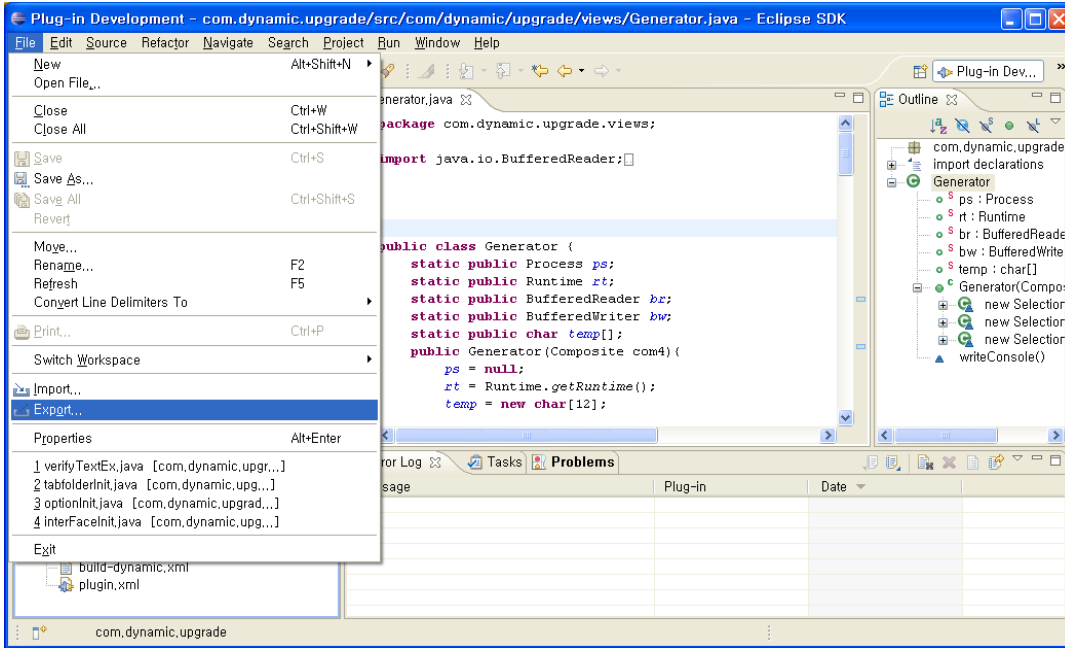
▶ 결과 화면



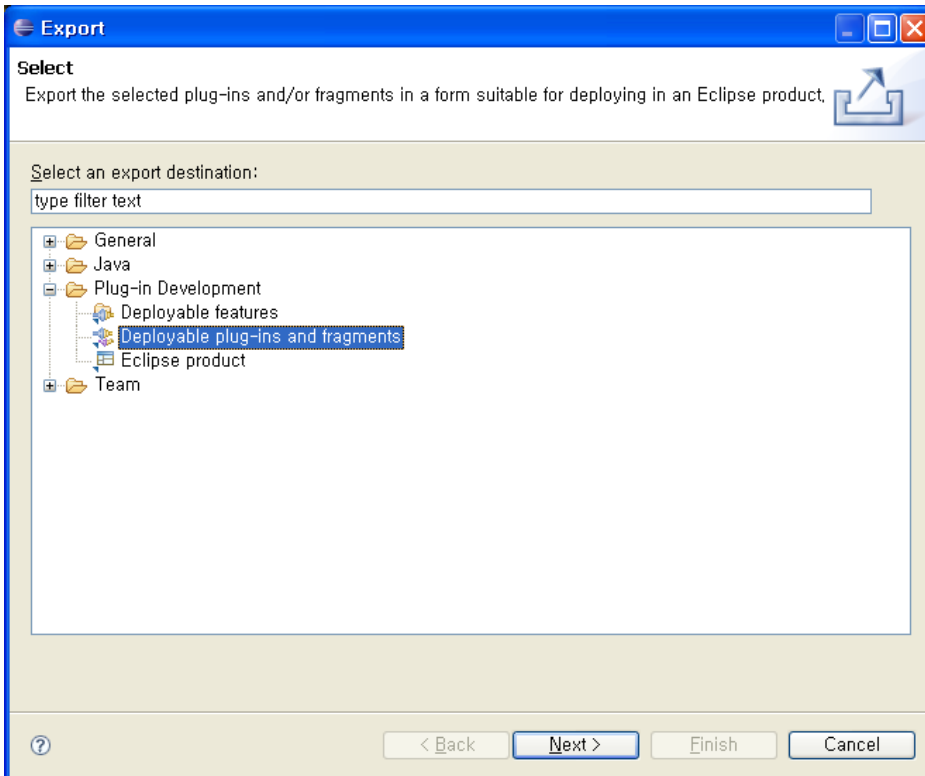
» 옵션을 선택하고 값을 넣은 뒤 **Calculation** 버튼을 클릭하면 아래 콘솔 창에 결과가 출력된다.

▶ **Export** 하는 방법은 아래와 같다.

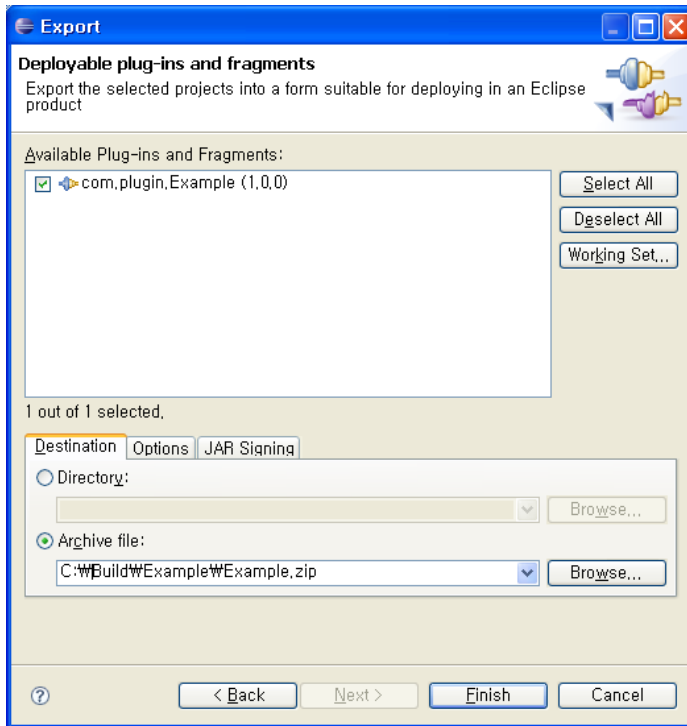
▶ **File -> Export**를 클릭한다.



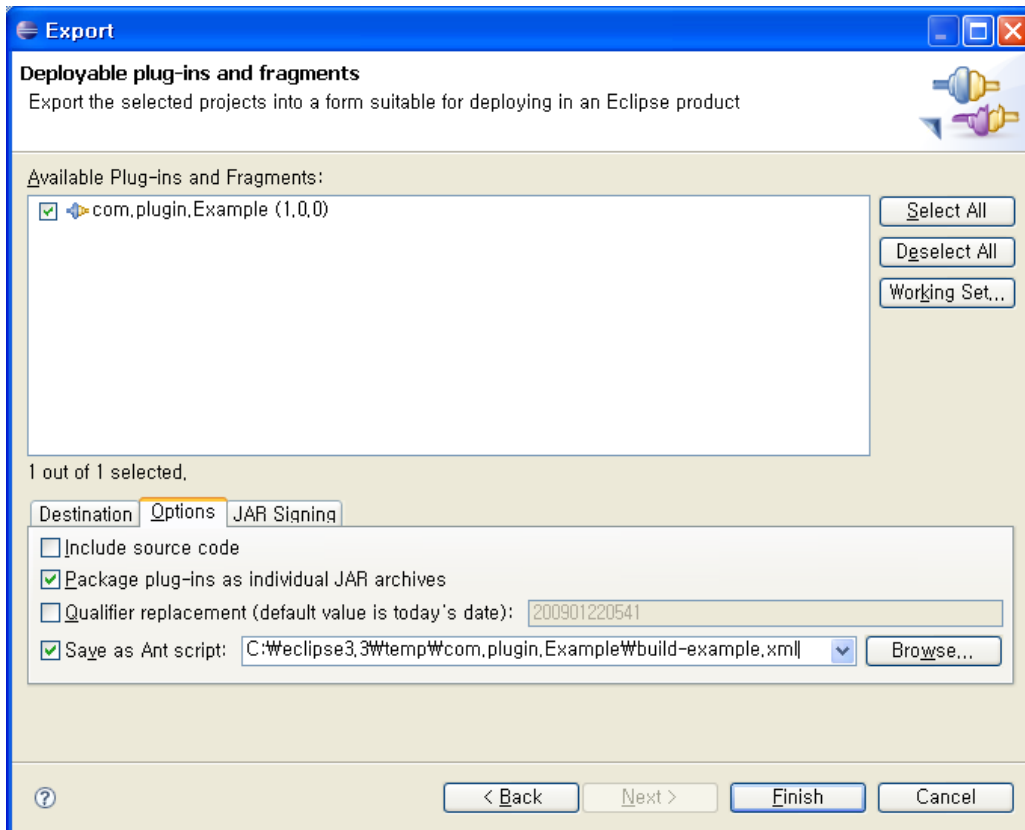
▶ **Deployable plug-ins and fragments**를 선택하고 **Next** 클릭



- ▶ 플러그인의 압축 파일을 만들 장소를 지정한다.



- ▶ 옵션을 정한다.

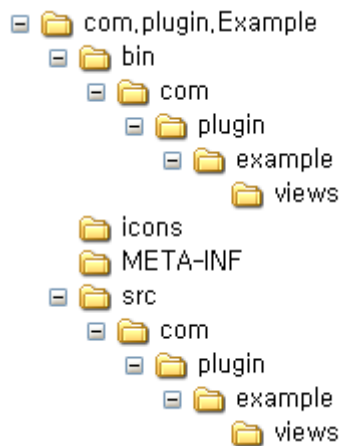




- ▶ 필요한 설정이 끝나면 **Finish**를 클릭하면 빌드되어 압축된 플러그인이 생성된다.
- ▶ 플러그인을 적용하기 위해 생성된 압축파일을 풀어 \*.jar 파일을 **Eclipse/plugins** 에 복사한 다음 이클립스를 실행하면 자동으로 적용된다.

## Part VI. Full Sources

- ▶ **Example** 프로젝트의 트리는 **workspace\com.plugin.Example**에 생성된다. 이 폴더 아래 **bin**, **icons**, **META-INF**, **src** 폴더가 있고, **.classpath** 파일과 **.project** 파일과 **build.properties** 파일과 **plugin.xml** 파일이 존재한다. **bin** 폴더에는 클래스들이 존재한다. **icons** 폴더에는 필요한 **icon**이 존재한다. **META-INF** 폴더에는 **MANIFEST.MF** 가 존재한다. **src** 폴더에는 **com\plugin\example** 아래에 **views** 폴더와 **Activator.java**가 존재하고, **views** 폴더에는 **ExampleView.java**와 **form.java**가 존재한다.



- ≫ 위의 그림이 프로젝트의 폴더트리를 간단하게 보여준다.

### ▶ Activator.java

```

package com.plugin.example;

import java.io.PrintStream;

import org.eclipse.jface.resource.ImageDescriptor;

import org.eclipse.ui.console.ConsolePlugin;

import org.eclipse.ui.console.IConsole;

import org.eclipse.ui.console.IConsoleManager;

import org.eclipse.ui.console.MessageConsole;

import org.eclipse.ui.console.MessageConsoleStream;

import org.eclipse.ui.plugin.AbstractUIPlugin;

```

```
import org.osgi.framework.BundleContext;

/**
 * The activator class controls the plug-in life cycle
 */
public class Activator extends AbstractUIPlugin {

    // The plug-in ID
    public static final String PLUGIN_ID = "com.plugin.Example";

    // The shared instance
    private static Activator plugin;

    /**
     * The constructor
     */
    public Activator() {
        MessageConsole console = new MessageConsole("Compile
Process", null);
        IConsoleManager manager =
(IConsoleManager) ConsolePlugin.getDefault().getConsoleManager();
        manager.addConsoles(new IConsole[]{console});
        manager.showConsoleView (console);
        MessageConsoleStream stream = console.newMessageStream();
        System.setOut(new PrintStream(stream));
    }

    /*
     * (non-Javadoc)
     * @see
    org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.Bundle
Context)
     */
    public void start(BundleContext context) throws Exception {
        super.start(context);
        plugin = this;
    }
}
```

```
    }

    /*
     * (non-Javadoc)
     * @see
     org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext)
    */
    public void stop(BundleContext context) throws Exception {
        plugin = null;
        super.stop(context);
    }

    /**
     * Returns the shared instance
     *
     * @return the shared instance
     */
    public static Activator getDefault() {
        return plugin;
    }

    /**
     * Returns an image descriptor for the image file at the given
     * plug-in relative path
     *
     * @param path the path
     * @return the image descriptor
     */
    public static ImageDescriptor getImageDescriptor(String path) {
        return imageDescriptorFromPlugin(PLUGIN_ID, path);
    }
}
```

## ► ExampleView.java

```
package com.plugin.example.views;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.*;

/**
 * This sample class demonstrates how to plug-in a new
 * workbench view. The view shows data obtained from the
 * model. The sample creates a dummy model on the fly,
 * but a real implementation would connect to the model
 * available either in this or another plug-in (e.g. the workspace).
 * The view is connected to the model using a content provider.
 * <p>
 * The view uses a label provider to define how model
 * objects should be presented in the view. Each
 * view can present the same model objects using
 * different labels and icons, if needed. Alternatively,
 * a single label provider can be shared between views
 * in order to ensure that objects of the same type are
 * presented in the same way everywhere.
 * <p>
 */

public class ExampleView extends ViewPart {

    /**
     * The content provider class is responsible for
     * providing objects to the view. It can wrap
     * existing objects in adapters or simply return
     * objects as-is. These objects may be sensitive
     * to the current input of the view, or ignore
```

```
* it and always show the same content
* (like Task List, for example).
*/
private form fr;

/**
 * The constructor.
 */
public ExampleView() {

}

/**
 * This is a callback that will allow us
 * to create the viewer and initialize it.
 */
public void createPartControl(Composite parent) {
    Composite cont = new Composite(parent, SWT.BORDER);
    // cont.setLayout(new FillLayout());
    fr = new form(cont);
}

@Override
public void setFocus() {
    // TODO Auto-generated method stub
}

/**
 * Passing the focus request to the viewer's control.
 */
}
```

## ► form.java

```
ackage com.plugin.example.views;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.List;
import org.eclipse.swt.widgets.Text;

public class form {
    static private Process ps;
    static private BufferedReader br;
    static private BufferedWriter bw;
    private Group startGroup;
    private Group executeGroup;
    final List list;
    private Button sum;
    private Button mult;
    final Text t1;
    final Text t2;
    private Button exe;

    public form(Composite shell){
        ps = null;

        startGroup = new Group(shell, SWT.NULL);
        startGroup.setText("Start Setting");
        startGroup.setBounds(25, 25, 300, 80);
```

```
list = new List(startGroup, SWT.SINGLE);
list.setItems(new String[] {
    "Integer", "Float"});
list.setBounds(25, 25, 50, 25);
list.setSelection(0);

executeGroup = new Group(shell, SWT.NULL);
executeGroup.setText("Execute");
executeGroup.setBounds(25, 120, 300, 100);

sum = new Button(executeGroup, SWT.RADIO);
mult = new Button(executeGroup, SWT.RADIO);
sum.setText("Addition");
mult.setText("Multiplication");
sum.setBounds(25, 25, 80, 25);
mult.setBounds(110, 25, 90, 25);

t1 = new Text(executeGroup, SWT.SINGLE);
t2 = new Text(executeGroup, SWT.SINGLE);
t1.setBounds(25, 60, 50, 15);
t2.setBounds(85, 60, 50, 15);

exe = new Button(executeGroup, SWT.PUSH);
exe.setText("Calculation");
exe.setBounds(150, 50, 90, 25);
exe.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        String str;
        try {
            if (list.getSelectionIndex() == 0) {
                str = "C://calculation.exe i";
            } else {
                str = "C://calculation.exe f";
            }
        }
    }
});
```

```
        }
        if (sum.getSelection()) {
            str = str+ " s "+t1.getText()+
"+t2.getText ();
        }else {
            str = str+ " m "+t1.getText()+
"+t2.getText ();
        }
        ps = Runtime.getRuntime().exec(str);
        br = new BufferedReader(new
InputStreamReader(ps.getInputStream()));
        bw = new BufferedWriter(new
OutputStreamWriter(ps.getOutputStream()));
        System.out.println(br.readLine());
    } catch(IOException e) {
        e.printStackTrace();
    }
    }
    });
}
}
```



---

► MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Example Plug-in
Bundle-SymbolicName: com.plugin.Example; singleton:=true
Bundle-Version: 1.0.0
Bundle-ClassPath: example.jar
Bundle-Activator: com.plugin.example.Activator
Bundle-Vendor: PLUGIN
Require-Bundle: org.eclipse.ui,
org.eclipse.core.runtime,
org.eclipse.ui.console
Eclipse-LazyStart: true
```

► plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

  <extension
    point="org.eclipse.ui.views">
    <category
      name="Example Category"
      id="com.plugin.Example">
    </category>
    <view
      name="Example View"
      icon="icons/sample.gif"
      category="com.plugin.Example"
      class="com.plugin.example.views.ExampleView"
      id="com.plugin.example.views.ExampleView">
    </view>
    </extension>

</plugin>
```

**► build.properties**

```
source.example.jar = src/  
output.example.jar = bin/  
bin.includes = plugin.xml,\n              META-INF/,\  
              example.jar,\  
              icons/
```

**► calculation.c**

```
#include <stdio.h>  
#include <stdlib.h>  
  
int type;  
  
void sum(int i1, int i2, float f1, float f2) {  
    if (type == 0) {  
        printf("%d + %d = %d\n", i1, i2, i1+i2 );  
    } else if(type == 1) {  
        printf("%f + %f = %f\n", f1, f2, f1+f2 );  
    }  
    return ;  
}  
  
void mul(int i1, int i2, float f1, float f2) {  
    if (type == 0) {  
        printf("%d * %d = %d\n", i1, i2, i1*i2 );  
    } else if(type == 1) {  
        printf("%f * %f = %f\n", f1, f2, f1*f2 );  
    }  
    return ;  
}  
  
void main(int argc, char* argv[]){  
    int i1, i2;  
    float f1, f2;
```

```

    if (argv[1][0] == 'i') {
        type = 0;

        i1 = atoi(argv[3]);
        i2 = atoi(argv[4]);
    } else if(argv[1][0] == 'f'){
        type = 1;

        f1 = atof(argv[3]);
        f2 = atof(argv[4]);
    }

    if ( argv[2][0] == 's') {
        sum(i1, i2, f1, f2);
    } else if (argv[2][0] == 'm'){
        mul(i1, i2, f1, f2);
    }

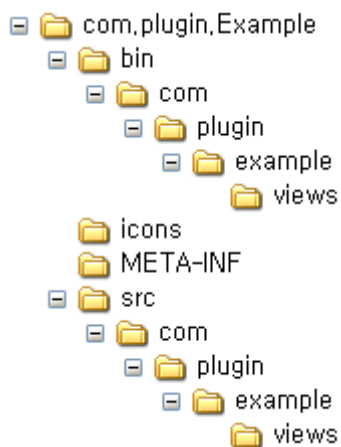
    return ;
}

```

▶ build 방법

Part II. Project에서 설명에 따라 프로젝트를 작성한다.

프로젝트 이름은 `com.plugin.Example`로 한다.



위와 같은 폴더트리로 프로젝트가 생성된다.

`com.plugin.Example` 폴더에 있는 `build.properties`, `plugin.xml`을 `full source`에 있는 내용으로 수정한다.  
`META-INF` 폴더에 있는 `MANIFEST.MF` 또한 `full source`의 내용으로 수정한다.

`src\com\plugin\example`에 존재하는 `Acitvator.java`를 `full source`의 내용으로 수정한다.

처음 프로젝트 생성할 때 생성된 `src\com\plugin\example\views` 에 존재하는 `SampleView.java` 파일은 이름을 수정하여 `ExampleView.java`로 수정하거나 `SampleView.java`를 삭제하고 `ExampleView.java`를 생성하여 `full source`의 내용으로 작성한다. 또한 `full source`의 `form.java`를 생성하여 `full source`의 내용으로 작성한다.

이제 이클립스를 실행하여 **F5** 키를 눌러서 새로 고침을 한다.

**Part V. Example** 에서 설명한 **Export** 하는 방법을 사용하여 작성된 프로젝트를 플러그인을 생성한다. `calculation.c`를 컴파일하여 `calculation.exe`로 생성하여 `C:\` 폴더 아래에 존재하게 한다.

**Export**가 완료되면 `C:\Build\Example` 아래에 `Example.zip` 가 생성된다. 이 파일의 압축을 풀어 `com.plugin.Example_1.0.0.jar` 파일을 `eclipse` 폴더 아래 `plugins`폴더에 존재하게 하고 이클립스를 실행시키면 작성된 플러그인이 해당 이클립스에서 구동하게 된다.